A Robust and Light Weight Authentication Framework for Hadoop File System in Cloud Computing Environment

Mrudula Sarvabhatla
NBKRIST
Nellore 524 413, A.P
mrudula.s911@gmail.com

M.Chandra Mouli Reddy
VelTech University, Chennai
mouli.veltech@gmail.com

Chandra Sekhar Vorugunti
Dhirubhai Ambani Institute of ICT
Gandhinagar-Gujarat.
vorugunti_chandra_sekhar@daiict.ac.in

ABSTRACT
The advancement of web and mobile technologies results in the rapid augmentation of traditional enterprise data, IoT generated data, social media data which outcomes in peta bytes and exa bytes of structured and un structured data across clusters of servers per day. The storage, processing, analyzing and securing these big data is becoming a serious concern to large and medium enterprises. Hadoop or HDFS is a distributed file system based on cloud for storage and processing of large voluminous amounts of data across clusters of servers. Along with huge potential for dynamism for processing and scalability, HDFS also brought inherent security drawbacks like lack of authentication and authorization of remote user connecting to cluster, missing of encryption of sensitive data at communication, storage and processing levels. These existing drawbacks demands for a robust, light weight security framework for HDFS. In this context, we propose a secure and light weight remote user authentication framework for HDFS, which guarantees all the critical security requirements of a distributed file system.

Categories and Subject Descriptors
D.4.6 [**Security and Protection**]: Authentication, Cryptographic controls.

General Terms
Hadoop, HDFS, Authentication

Keywords
Hadoop Framework, Hadoop Distributed File System, Authentication, Authorization, Big Data.

1. INTRODUCTION
The advancement of web, communication and sensor technologies results in the rapid augmentation of data from various sources like enterprise, Internet of Things, social media which results in 2.5 quintillion bytes of structured and un structured data per day [1]. The storage, processing, analyzing and securing these big data is becoming a serious concern to large and medium enterprises. To cater the enterprise data storage and processing needs, Apache developed Hadoop project which is a java based, open-source framework for distributed storage and processing of enterprise data. Clusters form heart of Hadoop file system, which stores peta bytes of data and have ability to scale according to dynamic processing needs of e-commerce enterprises.

WCI '15, August 10-13, 2015, Kochi, India
 ISBN 978-1-4503-3361-0/15/08…$15.00
DOI: http://dx.doi.org/10.1145/2791405.2791410

The e-commerce giants like facebook, amazon and search engine giants like google, yahoo are using hadoop to handle their dynamic data storage, retrieval and processing needs. Even though HDFS is best suited for data storage and processing needs, it suffers its lower in enterprise data security [2, 3,12]. The latest version of HDFS has very elementary implementation of security features [2,6,10,12], in which the access control mechanism is optional or advisory. As HDFS runs on top of UNIX operating system, hdfs relies on unix for user authentication, hdfs authenticates the user trying to login by querying the unix for 'whoami' command and based on the UNIX reply it authenticates the user [3].In hdfs, any user can connect directly to datanode by bypassing the namenode and can perform ‘store’, ‘read’ operations or execution of command [6].The same was demonstrated at the latest Cloudera's Hadoop Hackathon. Due tothese above discussed pitfalls in HDFS, the enterprises running on Hadoop are vulnerable to the following attacks [6]: Impersonation Attack: Unauthorized clients can mimic or imitate as authorized users and can access the datanodes of cluster.

Node or Cluster head Bypass Attack: The attacker can execute the command on the data blocks of datanodes by bypassing the cluster head or namenode.

Eavesdropping/ passive attack: The attacker can sniff the data packets exchanged between client and datanodes, due to plaintext transferring of the data.

Authentication in HDFS is a potential area, in which a slight quantity of end to end security analysis has been deliberated [2,3,6,13]. Also very few [4,5,7,8,9,11,13,14] authentication and authorization frameworks has been proposed by exploring various practices like Elliptic Curve Cryptography [12], Wireless Sensor Networks [15] etc., which are having their own drawbacks like using heavy weight cryptographic operations or vulnerable to cryptographic attacks. Rahul et al [1] have proposed an authentication frame work using heavy weight cryptographic operations like encryption/decryption etc., which are resource consuming and vulnerable to replay attack etc. Nivethitha et al [4] have proposed authentication framework based on one time pad using heavy weight cryptographic operations like encryption etc. Sadhasivam et al [5] have proposed an authentication frameworks using properties of medians of a triangle, which also consumes computation cost. Ibrahim et al [9] proposed authentication frameworks based on trusted computing.

In these next segments, we briefly discuss on existing hadoop architecture in section 2. In section 3, we analyze our proposed security framework for HDFS. In section4, we will analyze the security strengths of our proposed scheme. In section 5 study the cost and security analysis of our proposed scheme against various recently proposed schemes. In section 6, we conclude the manuscript.

2. PROPOSED SECURITY FRAMEWORK FOR HADOOP DISTRIBUTED FILE SYSTEM

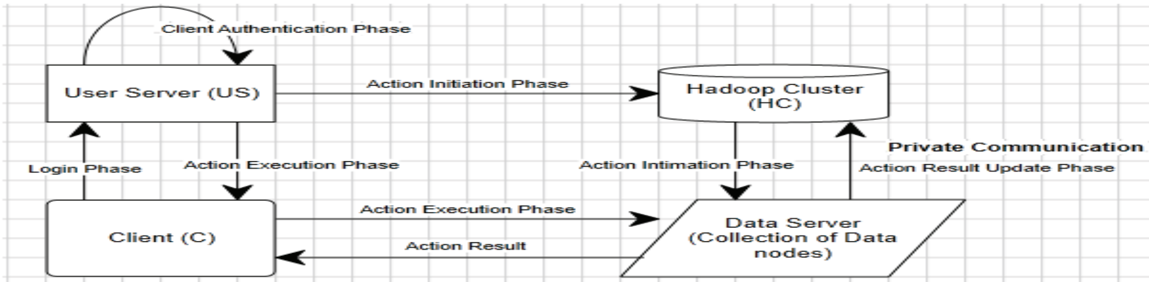

Figure 1.The above figure illustrates our proposed authentication and authorization framework for Hadoop/HDFS.

C_i: the i^{th}client, connecting to data node to complete the 'action'.
'action': action is a sub set of operations {read,write,cmd} which C_i executes on data node.
ID_i: the identity of C_i.
PW_i:the password of C_i.
US_j: the identity of user server, which acts as an interface to C_i and hadoop cluster.
CH_k: the K^{th} cluster head. (also Name node)
ID_{CHk}: Identity of cluster head CH_k
data_node_id: list of id's of nodes on which C_i can perform read or write or cmd execution.
K_{cus}: the secret key known to C_i and US_j.
K_{sch}:the secret key shared between US_j and CH_k.
K_{sdn}:the secret key shared between US_j and data node
X : the US_j long term secret key.
R_c :the arbitrary number chosen by US_j for C_i during registration stage.
R_{dn}:the arbitrary number chosen US_j for CH_k during 'action' initiation phase.
h(.) : a secure one-way and collision resistant hash function.
$\oplus$: the exclusive – OR (XOR) operation.
|| :Concatenation Operator.
U_i: A user. C_i or U_i can be used interchangeably.

2.1 The order of execution phases in our proposed scheme and communication entities involved in those phases are given below

a) Client Registration phase: $\mathbf{C_i}$ **- >$\mathbf{US_j}$**
b) Client Login phase: $\mathbf{C_i}$ **- >$\mathbf{US_j}$**
c) Client Authentication phase: $\mathbf{C_i}$ **- >$\mathbf{US_j}$**
d) 'Action' initiation phase: **$\mathbf{US_j}$ -> $\mathbf{CH_k}$**
e) 'Action' intimation phase: **$\mathbf{CH_k}$ -> data_node_id**
f) 'Action' execution phase:**$\mathbf{C_i}$-> data_node_id**

2.2 Pre Deployment Phase

Each cluster head is assigned with an identity ID_{CHk} 1<=k<=m where 'm' is the number of cluster heads. Each user server US_j and cluster head CH_k shares a symmetric secret key K_{sch}.

Each user server (US_j) and data node shares a symmetric secret key K_{sdn}.

2.3 Client Registration Phase

Client(C_i)(K_{cus}) **User Server(US_j)(K_{cus})**
$HID_i = h(ID_i||X)$ - >$K_i = K_{cus} \oplus h(ID_i \oplus X), A_i, R_i$

Select PW_i and an arbitrary number a_i
C_i computes: $APW_i = h(a_i \oplus PW_i)$
$HAPW_i = APW_i \oplus h(K_{cus} \oplus T1)$
$\{ID_i, HAPW_i, T1\}$

Computes:
$HID_i = h(ID_i||X)$
Index for $HID_i$to get K_i.
$K_{cus} = K_i \oplus h(ID_i \oplus X)$, $HK = h(K_{cus} \oplus T1)$
$APW_i = HAPW_i \oplus HK$
GenerateR_c
$V_i=h(ID_i||R_c||APW_i||X)$
$R_i= R_c \oplus h(ID_i||K_{cus}||APW_i)$
$K_i = K_{cus} \oplus h(ID_i||X)$
$A_i = APW_i \oplus h(X||K_{cus}||ID_i)$
US_j updates its data base entry for C_i and stores A_i, R_i.
$\{V_i, R_i, h(.)\}$

This phase is invoked whenever a client C_i needs to register with the user server (US_j) to access the hadoop resources.
(R1). C_i opts his password PW_i and an arbitrary number a_i. C_i computes masked password $APW_i = h(a_i \oplus PW_i)$, $HAPW_i = APW_i \oplus h(K_{cus} \oplus T1)$and submits the registration request $\{ID_i, HAPW_i, T1\}$ to US_j.
(R2) On receiving the registration request $\{ID_i, HAPW_i, T1\}$, US_j computes $HID_i = h(ID_i||X)$ and indexes its data base for HID_i and retrieves K_i(which is already stored during pre-initialization phase). C_i intercepts K_{cus} from K_i i.e $K_{cus} = K_i \oplus HID_i$ and computes $HK = h(K_{cus} \oplus T1)$.$K_{cus}$ is the secret key shared between C_i and US_j during pre-initialization phase.
(R3) US_j retrives APW_i from $APW_i = HAPW_i \oplus HK$. US_j generates a random number R_c, and computes $V_i=h(ID_i||R_c||APW_i||X)$,$R_i= R_c \oplus h(ID_i||K_{cus}||APW_i)$.
(R4) To authenticate the user for future logins, the US_j stores the variables $A_i = APW_i \oplus h(X||K_{cus}||ID_i)$, $R_i= R_c \oplus h(ID_i||K_{cus}||APW_i)$ in its data base against the index $h(ID_i||X)$. i.e $H(ID_i||X)$- >$K_i = K_{cus} \oplus h(ID_i \oplus X), A_i, R_i$.
(R5) The US_jforwards the values $\{V_i, R_i, h(.)\}$ through a secure communication channel to U_i.

2.4 Client Login Phase:

Client(C_i)(K_{cus}) **User Server(US_j)(K_{cus})**
($HID_i = h(ID_i||X)$ - > K_i, A_i, R_i)

Computes:
$RAPW_i = h(R_c \oplus K_{cus} \oplus APW_i \oplus T2)$,
$M1=(action,V_i) \oplus RAPW_i$

$\xrightarrow{\{ID_i, M1, T2\}}$

Whenever the registered client C_i is in need of data storage or data access or execution of a command through HDFS, C_i can login to the HDFS system via user server (US_j) as follows:

(L1)The client C_i computes $RAPW_i = h(R_c \oplus K_{cus} \oplus APW_i \oplus T2)$ using the secret key shared between C_i and US_j i.e K_{cus}, the random number assigned by US_j to C_i i.e R_c, APW_i and the current time T2.

(L2) The client C_i computes $M_i=(action,V_i) \oplus RAPW_i$, and submits the login request $\{ID_i, M1, T2\}$ at time T2 to user server US_j.

2.5 Client Authentication Phase:

Client(C_i)(K_{cus}) **User Server(US_j)(K_{cus})**
($HID_i = h(ID_i||X)$ - > K_i, A_i, R_i)

$\xrightarrow{\{ID_i, M1, T2\}}$

Receive the request at time $T2^*$
Verify: $(T2^*-T2) \le \Delta t$,
Compute: $HID_i = h(ID_i||X)$,as the server knows his secret key 'X'
Retrieve A_i, R_i, K_i
$K_{cus}^* = K_i \oplus h(ID_i||X)$
$APW_i^* = A_i \oplus h(X||K_{cus}||ID_i)$
$R_c^* = R_i \oplus h(ID_i||K_{cus}||APW_i)$
$V_i^* = h(ID_i^*||R_c^*||APW_i^*||X)$
$RAPW_i^* = h(R_c^* \oplus K_{cus}^* \oplus APW_i^* \oplus T2)$
$(action,V_i) = M_i \oplus RAPW_i^*$.
Verify V_i^* (computed) = V_i (received).
If Yes, C_i is authenticated.

(A1) On receiving the login request from C_i at time $T2^*$, the US_j validates the time of reception i.e $(T2^*-T2) \le \Delta t$, on validating the time interval, US_j proceeds to compute $HID_i = h(ID_i||X)$ and indexes the data base for HID_i and retrieves the stored values A_i, R_i, K_i.

(A2) On retrieving A_i, R_i, K_i, US_j retrieves $K_{cus}^* = K_i \oplus h(ID_i||X)$, $APW_i^* = A_i \oplus h(X||K_{cus}||ID_i)$, $R_c^* = R_i \oplus h(ID_i||K_{cus}||APW_i)$ and computes $V_i^* = h(ID_i^*||R_c^*||APW_i^*||X)$.

(A3)US_j computes $RAPW_i^* = h(R_c^* \oplus K_{cus}^* \oplus APW_i^* \oplus T2)$ and retrieves $(action,V_i) = M_i \oplus RAPW_i^*$.

(A4) US_j compares the retrieved V_i with the computed V_i^*. If both are equal, the client C_i is authenticated by US_j. On authenticating the client C_i, US_j must contact the dataserver via name node or cluster head to execute the 'action' task required by C_i. To achieve the same, US_j performs the subsequent steps:

2.6 'Action' Initiation Phase:

User Server (US_j)(K_{sch}) **Cluster Head(CH_k) (K_{sch})**
(ID_{us}, R_{dn}, Status)

Generate a random number R_{dn}
$R_{SCH} = h(K_{sch} \oplus ID_{us})$,
$MUS_i=(action|| R_c|| R_{dn}||ID_i||ID_{us}) \oplus R_{SCH}$

$\xrightarrow{\{ID_i, ID_{us}, MUS_i, R_{dn}\}}$

Retrieve K_{sch} based on ID_{us}.
Compute: $R_{SCH} = h(K_{sch}||ID_{us})$
$MUS_i \oplus R_{SCH}=(action|| R_c|| R_{dn}||ID_i||ID_{us})$
Retrieve the stored R_{dn}, status if any, based on ID_{us}.
If Status =1,reject the request from CH_k.
if there is no entry for ID_{us}, make an entry <ID_{us}, R_{dn}, Status=0> in the data base
// Names node checks the meta data and collects the list of data nodes useful for successful execution of 'action' request by C_i.//
$MCH_i = (data\_node\_id||ID_i||R_{dn}||ID_{us}) \oplus h(R_{dn}||K_{sch}||R_c)$

$\xleftarrow{\{ID_{us}, MCH_i\}}$

Retrieves
$MCH_i \oplus h(R_{dn}||K_{sch}) = (data\_node\_id||ID_i||R_{dn}||ID_{us})$

The user server US_j connects to the name node or cluster head (CH_k) and submits the data request raised by the client C_i.

(AI1) US_j generates a random number R_{dn} and computes $R_{SCH} = h(K_{sch} \oplus ID_i)$, $MUS_i = (action|| R_c||R_{dn}||ID_i||ID_{us}) \oplus R_{SCH}$.

(AI2) US_j submits the action request$\{ID_i, ID_{us}, MUS_i, R_{dn}\}$ to the cluster head (CH_k).

(AI3) On receiving the action request from US_j, based on ID_{us}, CH_k retrieves the secret key K_{sch} which is shared between US_j and CH_k.

(AI4) CH_k computes $K_{SCH} = h(K_{sch}||ID_i)$ and retrieves $(action|| R_c|| R_{dn}||ID_i||ID_{us})$ from MUS_i. i.e $MUS_i \oplus K_{SCH} = (action||R_c|| R_{dn}|| ID_i||ID_{us})$.

(AI5) Based on ID_{us} and R_{dn} combination, CH_k checks the database entry for ID_{us} and R_{dn} combination. If any entry is found, CH_k checks the 'Status' flag, if it is 1, then CH_k confirms that it is a replay message and rejects the request, else proceeds further.

(AI6) Based on the 'action' field, the name node checks the meta data and finalize the list of data nodes i.e data_node_id is required to process the 'action' task.

(AI7) CH_k computes $MCH_i = (data\_node\_id||ID_i||R_{dn}|| ID_{us}) \oplus h(R_{dn}||K_{sch})$ and reply back to US_j with the message $\{ID_{us}, MCH_i\}$.

(AI8) On receiving the reply message from the cluster head CH_k, US_j computes $MCH_i \oplus h(R_{dn}||K_{sch}) = (data\_node\_id||ID_i||R_{dn}||ID_{us})$ to retrieve the list of data nodes i.e data_node_id. US_j validates the request by cross checking the arbitrary number R_{dn} and its id ID_{us}.

On intercepting the data_node_id, US_j performs a sequence of steps which are in lined below.

2.7 'Action' intimation phase:

Name Node or Cluster Head(CH_k) (K_{chs}) **Data Node**

$\xrightarrow{(ID_i|| R_{dn}|| R_c||action||data\_node\_id)}$

On finalizing the data node list for the 'action' request made by US_j, the cluster head updates the datanodes in the data list with a message $(ID_i|| R_{dn}|| R_c||action||data\_node\_id)$.Data node intercepts the message to get ID_i, R_{dn}, R_c, action, data_node_id. The message from CH_k, instructs the data nodes that, a client with identity ID_i and chosen random numbers R_{dn}, $R_c$will contact to execute 'action' task.

2.8 'Action' execution phase:

(AE1)US_j generates the session key $S.K_{UD}$ which is to be used to exchange the data securely between C_i and data_node_id.

(AE2) US_j computes $RSK = h(R_{dn}||S.K_{UD})$ where R_{dn} is the random number used to frame the action request $\{ID_i, ID_{us}, MUS_i, R_{dn}\}$ which US_j submitted to the cluster head (CH_k).

(AE3) US_j computes $M1 = (ID_i||R_{dn}||data\_node\_id||\ S.K_{UD}||R_c) \oplus h(R_{dn}||K_{udn})$, $M2 = (ID_i||R_c||R_{dn}||S.K_{UD}||data\_node\_id) \oplus h(R_c||K_{cus})$ and submits the reply {M1,M2} to the client C_i. M1 is meant for data node and M2 is for C_i.

(AE4) On receiving the reply from US_j, C_i computes $HK_{CUS} = h(R_c||\ K_{cus})$ (as the C_i knows both R_c and K_{cus}) and verifies whether the received R_c is equal to R_c it received from US_j in login request. If both are same, then U_i authenticates US_j and proceeds further.

(AE5) C_i intercepts ID_i, R_c, R_{dn},$S.K_{UD}$,data_node_id from $M2\oplus HK_{CUS}$ and computes $M3 = (ID_i||R_{dn}||S.K_{UD}||data\_node\_id) \oplus h(S.K_{UD}||\ R_{dn})$.

(AE6) C_i submits the action execution request $\{ID_i, M1, M3\}$ to data server/data node.

(AE7) On receiving the action execution request from C_i, the data node retrieves the list of data nodes required to execute the 'action' request raised by C_ii,.e data_node_id,R_{dn}, Status from D.B.

(AE8) if the 'Status' flag is set to 1, then the request was processed earlier and the data node rejects the request, else proceeds further.

(AE9) Data node proceeds to compute $T1 = h(R_{dn}||K_{udn})$, $M1\oplus T1 = ID_i^*, R_{dn}^*, data\_node\_id^*, S.K_{UD}^*, R_c^*$. Data node compares ID_i^*, R_{dn}^*, R_c^*, action*received from C_i against ID_i, R_{dn}, R_c, action received from CH_k. If all values are valid, then data_node validates the 'action' request.

(AE10) Data node proceeds to compute $T2 = h(S.K_{UD}||\ R_{dn})$, $M3\oplus T2 = ID_i^{\#}, R_{dn}^{\#}, S.K_{UD}^{\#}, data\_node\_id^{\#}$. Data node compares if $ID_i^* = ID_i^{\#}$ and $R_{dn}^* = R_{dn}^{\#}$ and $data\_node\_id^* = data\_node\_id^{\#}$ and $S.K_{UD}^* = S.K_{UD}^{\#}$ then data node authenticates C_i and US_i.

(AE11) Once the C_i is authenticated by data node, the data node proceeds to compute $T3 = h(S.K_{UD}||R_{dn})$ and $M4 = (R_c^*+1||\ data\_node\_id^*) \oplus\ T3$ and responds back to C_i with the message $\{ID_i, M4\}$.

(AE12) On receiving the reply message from the data node, C_i computes $T4 = h(S.K_{UD}||R_{dn})^*$

$M4\oplus T4 = R_c^*+1$, $data\_node\_id^*$, U_i validates R_c^* and $data\_node\_id^*$. if both are valid U_i authenticates data node.

(AE13) if the action is 'retrieve', then the data node retrieves the data from its blocks and reply back to C_i i.e data_retrieved $\oplus h(S.K)$.

(AE14) if the action is 'store' the data node stores the data in its data blocks and reply back the status of store operation i.e success or failure.(Success) $\oplus$ h(S.K) else ('Fail) $\oplus$ h(S.K).

(AE15) On successful completion of 'action' request, data node updates the 'Status' flag to 1, as the 'action' request from C_i corresponding to the arbitrary number is R_{dn} is successfully done.

User Server (US_i) | **U_i** | **Data Node**

(K_{sdn},K_{cus}) | **(K_{cus})** | **(K_{sdn})**(data_node_id,ID_i,R_{dn},R_c,action, Status)

Generate session key between U_i and data_node_id i.e $S.K_{UD}$

$RSK = h(R_{dn}||S.K_{UD})$

$M1 = (ID_i||R_{dn}||data\_node\_id||S.K_{UD}||R_c||action) \oplus h(R_{dn}||Ks_{dn})$

$M2 = (ID_i||R_c||R_{dn}||S.K_{UD}||data\_node\_id||action) \oplus h(R_c||\ K_{cus})$

$\xrightarrow{\{M1,M2\}}$

Compute: $h(R_c||\ K_{cus})$

$M2\oplus h(R_c||K_{cus}) = ID_i, R_c, R_{dn}, S.K_{UD}, data\_node\_id$

$M3 = (ID_i||R_{dn}||S.K_{UD}||data\_node\_id) \oplus h(S.K_{UD}||R_{dn})$.

$\xrightarrow{\{ID_i,M1,M3\}}$

Based on ID_i, retrieve data_node_id, R_{dn} from D.B.

Compute:

$DN1 = h(R_{dn}||Ks_{dn})$

$M1\oplus DN1 = ID_i^*, R_{dn}^*, data\_node\_id^*, S.K_{UD}^*, R_c^*, action^*$

Based on ID_i, data node fetches Status flag. If it is 1, then it infers that the request is replayed one and data node rejects the request, else proceed further

$DN2 = h(S.K_{UD}||\ R_{dn})$.

data node compares ID_i,R_{dn},R_c which is received from CH_k with the values sent by US_j i.e $ID_i^* = ID_i$, $R_{dn}^* = R_{dn}$, $R_c^* = R_c$, $action^* = action$.

If yes, data_node authenticates US_j.

$M3\oplus DN2 = ID_i^{\#}, R_{dn}^{\#}, S.K_{UD}^{\#}, data\_node\_id^{\#}, action^{\#}$

If $ID_i^* = ID_i^{\#}$ and $R_{dn}^* = R_{dn}^{\#}$ and $data\_node\_id^* = data\_node\_id^{\#}$ and $S.K_{UD}^* = S.K_{UD}^{\#}$ then data node authenticates C_i.

If C_i is authenticated, data_node updates Status flag to 1.

$T3 = h(S.K_{UD}||R_{dn})$

$M4 = (R_c^*+1||data\_node\_id^*) \oplus\ T3$

$\xleftarrow{\{ID_i,M4\}}$

Computes:

$T4 = h(S.K_{UD}||R_{dn})^*$, $M4\oplus T4 = R_c^*+1$, $data\_node\_id^*$

C_i validates R_c^* and $data\_node\_id^*$

(AE16) As shown in the above diagram, on successful completion of 'action' execute request by C_i, datanode updates the same to CH_k as a reply to 'action' intimation message i.e phase i.e (ID_i|| R_{dn}|| R_c|| action||data_node_id). The reply is (ID_i||R_{dn}|| R_c||action|| data_node_id||Status). On receiving the message from the data node, CH_k updates the 'Status' value against ID_{us}, R_{dn} combination to resist the reply attacks.

3. SECURITY ANALYSIS OF IMPROVED SCHEME

3.1 Counter to User Impersonation Attack (Authentication)

To mimic as a legal user C_i to user server US_j, the attacker 'E' must frame the valid login request{ID_i, M1, T2} where M1=(action,V_i)$\oplus$$RAPW_i$, $RAPW_i$ = h($R_c \oplus K_{cus} \oplus$ h($a_i \oplus PW_i$) $\oplus$T2). To compute M1, 'E' prerequisite K_{cus}, R_c, a_i, PW_i. As discussed in [7,8]. It is not possible to guess four unknown values in polynomial time by an attacker. (An attacker can guess only password, assuming it is a low entropy one. Apart from password PW_i, it is not possible to guess any value which is more than 16 bits). Hence 'E' cannot reframe a valid login message to impersonate C_i. In second scenario, during 'action' execution phase, as shown in the fig1, C_i sends {ID_i, M1,M3} to data_node_id. To frame a valid M1, M3, the attacker 'E' requires $S.K_{UD}$, which is not possible to intercept or guess by an attacker. Hence, our scheme successfully resists user impersonation attack.

Table 1.The equations accessible to attackers and the values known and unknown to them.

Equation	Equations in full form	Variables Unknown	Variables known
{ID_i,M1, T2}	M1=(action,V_i)$\oplus$$RAPW_i$ $RAPW_i$ = h($R_c \oplus K_{cus} \oplus$ h($a_i \oplus PW_i$) $\oplus$T2)	K_{cus},R_c, a_i, PW_i, action,V_i	T2,ID_i
{ID_i,ID_{us},$MUS_{i,\ dn}$}	MUS_i = (action|| R_c|| R_{dn}||ID_i||ID_{us})$\oplus$ R_{SCH} R_{SCH} = h($K_{sch} \oplus ID_i$)	K_{sch}, action, R_c	ID_i,ID_{us}, R_{dn}
{ID_{us}, MCH_i}	MCH_i = (data_node_id||ID_i||R_{dn})$\oplus$h(R_{dn}||K_{sch} ||R_c)	data_node_id, K_{sch}, R_c.	ID_i, R_{dn}
{M1, M2}	M1 = (ID_i||R_{dn} ||data_node_id||$S.K_{UD}$|| R_c|| action)$\oplus$h(R_{dn}||Ks_{dn}) M2 = (ID_i||R_c||R_{dn}||$S.K_{UD}$||data_node_id|| action) $\oplus$h(R_c|| K_{cus})	data_node_id, $S.K_{UD}$, R_c, action, K_{sdn}, K_{cus}	ID_i, R_{dn}
{ID_i, M1,M3}	M3 = (ID_i||R_{dn}||$S.K_{UD}$||data_node_id) $\oplus$h($S.K_{UD}$||R_{dn}).	$S.K_{UD}$, data_node_id	ID_i, R_{dn}
{ID_i, M4}	M4 = (R_c^*+1||data_node_id*)$\oplus$ T3	R_c, data_node_id, T3	None

3.2 Counter to Replay Attack (Authentication)

An attacker 'E' wish to replay a login request message sent by U_i to data node, 'E' can intercept the message from C_i i.e {ID_i, M1,M3} and forward the same to data node. As discussed in (AE16) of 'action' execution phase, on successful completion of 'action' request of C_i, based on ID_i and R_{dn}, the data node updates the 'Status' flag related to C_i and R_{dn} combination to 1. If the data node receives the replay messages, it retrieves 'Status' flag values based on ID_i and R_{dn} combination. If Status flag is 1, it implies that the request is replayed one and already processed. Hence, data node rejects the 'action' execution request from C_i. Therefore, it is concluded that our scheme is resistant to replay attacks.

3.3 Counter to Server Masquerade Attack

To deceive as a user server US_j, the attacker 'E' must send a message {M1,M2} where M1 = (ID_i||R_{dn}||data_node_id|| $S.K_{UD}$||R_c||action)$\oplus$h(R_{dn}||Ks_{dn}), M2 = (ID_i||R_c||R_{dn}||$S.K_{UD}$|| data_node_id||action) $\oplus$h(R_c|| K_{cus}) to C_i. As shown in the table 1, the attacker 'E' must know the values i.e data_node_id, $S.K_{UD}$, R_c, action, Ks_{dn}, K_{cus} to successfully frame valid M1, M2. It is computationally infeasible to guess or compute or intercept data_node_id, $S.K_{UD}$, R_c, action, Ks_{dn}, K_{cus} successfully. Hence, we can conclude that our scheme resists server masquerade attack.

3.4 Counter to Password Guessing attack

The only scope for an attacker 'E' to get PW_i of C_i is the login request {ID_i,M_i, T2} where M1=(action,V_i)$\oplus$$RAPW_i$ and $RAPW_i$ = h($R_c \oplus K_{cus} \oplus$ h($a_i \oplus PW_i$) $\oplus$T2). As shown in table 1, 'E' must know K_{cus}, R_c, a_i, action,V_i to perform guessing attack on unknown PW_i. It is impossible for 'E' to guess or intercept the above mentioned values. Hence, we can conclude that our scheme resists password guessing attack.

3.5 Counter to Stolen Verifier Attack

In our scheme, the user server US_j stores the PW_i in ahashed format i.e A_i = $APW_i$$\oplus$h(X||$K_{cus}$||$ID_i$) = h($a_i \oplus PW_i$)$\oplus$ h(X||K_{cus}|| ID_i) not as a plain text. Hence, it is impossible for an insider to compute the C_i password. If an insider steals the password verifier A_i, to compute APW_i, 'E' requires h(X||K_{cus}||ID_i) where 'X' is the server long term key, K_{cus} is the symmetric key shared between C_i and US_j. It is computationally infeasible for 'E' to compute or intercept these values. Therefore, we can confirm that our scheme is resistant to stolen verifier attack.

3.6 Achieves Strong Mutual Authentication

As discussed above, each communicating party i.e C_i, US_j and datanode authenticates each other on receiving the message from the corresponding entities. In (A4) of user authentication phase, US_j authenticates C_i. In (AI5) of action initiation phase, CH_k authenticates US_j. In (AI8) of action initiation phase, US_j authenticates CH_k. In (AE4) of action execution phase, C_i authenticates US_j. In (AE9) and (AE10) datanode authenticates C_i and US_j. In (AE13), C_i authenticates data node. Hence, in our scheme, all the communicating entities will authenticate each other. Therefore we can conclude that our scheme provides strong mutual authentication.

3.7 Counters Namenode bypass attack (Authorization)

As discussed in (AE9) of 'Action' execution phase, to process any 'action' request from C_i, the data node cross checks the action request from $C_i$with the data it received from CH_k.If both are valid, then only the data node proceeds. If there is no 'action' intimation message from CH_k to data node corresponding to ($ID_i||R_{dn}||R_c||$action) combination, data node rejects the request. If C_i or attacker replays the message, as discussed in (AE14), based on 'Status' flag, data node rejects the request. Hence, in our framework it is impossible for an attacker or legal user to bypass the name node.

3.8 Counters Eavesdropping/Passive attack

As discussed above, all the communication messages exchanged in our framework are XORed with hash value. As shown in the table 1, the hash values can be computed only by the intended recipients and impossible for attackers. Hence, even though the attackers sniff the data, he cannot retrieve the actual content or modify it. In our scheme, the communication messages are designed such that, the attacker must guess or intercept more than one value which are 128 bits length to compute one unknown value. This is impossible in linear polynomial time [7,8]. Hence, we can conclude that, our framework resists all major cryptographic attacks.

4. COST and SECURITY ANALYSIS

In this section we analyze communication and computation cost required by our framework and we compare the same with recently proposed security framework by Rahul et al [1]. Similar to widely accepted [7], we are also considering the identity, random/arbitrary numbers, symmetric keys and time stamps are of 128 bits.i.e ID_i,R_c, a_i,R_{dn}, the keys i.e K_{cus},K_{sch},$K_{sdn,}$ the timestamps T1,T2,T3are 128 bits long. The output of hash function is 128-bit.H, EX, E,D denote the time complexity for hash function, exponential operation, symmetric key encryption and decryption respectively. The communication cost in client login stage = {ID_i,M1, T2} = 128+128+128 = 384 bits. The communication cost in 'action' initiation phase= {ID_i,ID_{us},MUS_i, R_{dn}},{ID_{us}, MCH_i} = {128+128+128+128}, {128+128} = 768 bits.The communication cost in 'action' execution phase = {M1,M2}, {ID_i, M1,M3},{ID_i, M4} = {128+128},{128+128+ 128},{128+128} = 896. Total communication cost = 384 + 768+896 = 2048 bits. The cost comparison of the proposed user authentication and authorization framework for HDFS with very recently proposed Rahul et al scheme our framework requires only 14H compared to 101H by Rahul et al framework. Our framework reduced almost 87H operations. Similarly, the communication cost also reduced drastically.

5. CONCLUSION

In this manuscript, we have proposed first of its kind of secure and light weight authentication and authorization framework for HDFS. The proposed framework resolves the existing security issues in HDFS like authentication, authorization and name node bypassing etc, which are critial for outsourcing the sensitive enterprise data to cloud. One more positive side of our framework is that the computation and communication cost are negligible, as we have used only light weight hash and XOR operations,compared to heavy weight encryption, exponentiation operations as in similar works discussed.

6. REFERENCES

[1] Rahul ,P.K. GireeshKumar T.2015. A Novel Authentication Framework for Hadoop, *International conference on Intelligence and Evolutionary Algorithms in Engineering Systems (ICAEES 201)* , Vol 324, 2015, pp 333-340 .

[2] Venkata, N. I., Sailaja, A., and Srinivasa Rao, R. security issues associated with big data in cloud computing, International Journal of Network Security & Its Applications (IJNSA), Vol.6, No.3, May 2014.

[3] Devaraj, D., Owen, O., 'Malley.,Sanjay, R., Kan., Z., and, Adding security to Apache Hadoop, Horton works Technical report1, http://hortonworks.com/wp-content/uploads/2011/10/security-design_withCover-1.pdf.

[4] Nivethitha, S., Gangaa A., and Shankar ,S. Authentication Service in Hadoop Using one Time Pad, Indian Journal of Science & Technology ,vol 7, pp 56-62, Apr 2014.

[5] Sadasivam G.S.,Kumari, K..A., and Rubika S.*A Novel Authentication Service for Hadoop in Cloud Environment, IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pp 1-6, oct 2012,India.

[6] Jam, M.R, Khanli, L.M., Javan M.S., andAkbari, M.K. A survey on security of Hadoop*, International eConference on Computer and Knowledge Engineering (ICCKE),*pp: 716 - 721,29-30 Oct. 2014, India.

[7] Sandeep, K.S. "An Improved and Secure Smart Card Based Dynamic Identity Authentication Protocol", International Journal of Network Security, Vol.14, PP.39-46, Jan. 2012.

[8] Zhou,Quan.,Deqin, Xiao.,Tang. H., and Chunming R. TSHC: Trusted Scheme for Hadoop Cluster,Fourth *International Conference on Emerging Intelligent Data and Web Technologies* (EIDWT),9-11 Sept. 2013.

[9] Ibrahim, L., and Ning, Zhang., MapReduce: MR Model Abstraction for Future Security Study, *In Proc of the 7th International Conference on Security of Information and Networks,* SIN '14, 9-11 Sep, Univ of Glasgow, 2014.

[10] Yoon, S., J.,and Yong,T., K. A token-based authentication security scheme for Hadoop distributed file system using elliptic curve cryptography, springer journal of Journal of Computer Virology and Hacking Techniques, February 2015.

[11] James, D. and Ning, Zhang. Security issues relating to inadequate authentication in map reduce applications. In High Performance Computing and Simulation (HPCS), 2013 International Conference on, pages 281--288. IEEE, 2013.

[12] Yoon, S.,Jeong, Y., T., K.. A token-based authentication security scheme for Hadoop distributed file system using elliptic curve cryptography. springer journal of Journal of Computer Virology and Hacking Techniques.

[13] Bhushan, L. Open Source Authentication in Hadoop, Springer journal of Practical Hadoop Security ,2014, pp 51-74

[14] Ibrahim, L., and Ning, Z. SIN '14 Proceedings of the 7th International Conference on Security of Information and Networks,MapReduce: MR Model Abstraction for Future Security Study.

[15] Byoung,J.B,Young,J.K.,Young,K.Kim.,Ok,K.Ha., and Yong,K.An Intrusive Analyzer for Hadoop Systems Based on Wireless Sensor Networks International Journal of Distributed Sensor Networks, Volume 2014.